[SMART -- Tutorials -- JavaScript (smarthealthit.org)](https://docs.smarthealthit.org/tutorials/javascript/) https://docs.smarthealthit.org/tutorials/javascript/

Tutorial - Building a JavaScript App

Getting started

The SMART on FHIR JavaScript client library helps you build browser-based SMART apps that interact with a FHIR REST API server. It can help your app get authorization tokens, provide information about the user and patient record in context, and issue API calls to fetch clinical data.

This tutorial describes how to create a basic client-side JavaScript SMART app with no additional libraries and frameworks.

To get started with the SMART on FHIR JavaScript client library, you’ll need to:

Step 1. Include a script tag

Include a script tag referencing the library. The latest code is always available from the NPM CDN at <https://cdn.jsdelivr.net/npm/fhirclient/build/fhir-client.js>. Including this script will create a global FHIR object for you to work with. **Example:**

<script src="https://cdn.jsdelivr.net/npm/fhirclient/build/fhir-client.js"></script>

Step 2. Test against an open FHIR server

Now that you have the client library included, it might be a good idea to play with it a little bit and see what it is capable of. This is easy to do if we use an open FHIR server that does not require authentication.

Here is something basic to start with:

**Connect to an open server and browse patients**

const client = FHIR.client("https://r3.smarthealthit.org");

client.request("Patient").then(console.log).catch(console.error);

Once this is working you can try [other examples](http://docs.smarthealthit.org/client-js/request.html) and then proceed to the next step.

Step 3. SMART Authorization

The most common type of SMART app is designed to run within the EHR. Such an app must support the [EHR Launch flow](http://www.hl7.org/fhir/smart-app-launch/#ehr-launch-sequence). To do that, we need to separate our logic in two pages:

1. launch.html - to be called by the EHR to start the authorization process.
2. index.html - Upon successful authorization the EHR will redirect us there. This is where the actual app is initialized.

Step 3.1. Create the launch.html page

This is very simple blank page with one purpose - to launch the app:

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<title>Launch My APP</title>

<script src="https://cdn.jsdelivr.net/npm/fhirclient/build/fhir-client.js"></script>

</head>

<body>

<script>

FHIR.oauth2.authorize({

// The client\_id that you should have obtained after registering a client at

// the EHR.

clientId: "my\_web\_app",

// The scopes that you request from the EHR. In this case we want to:

// launch - Get the launch context

// openid & fhirUser - Get the current user

// patient/\*.read - Read patient data

scope: "launch openid fhirUser patient/\*.read",

// Typically, if your redirectUri points to the root of the current directory

// (where the launchUri is), you can omit this option because the default value is

// ".". However, some servers do not support directory indexes so "." and "./"

// will not automatically map to the "index.html" file in that directory.

redirectUri: "index.html"

});

</script>

</body>

</html>

Note that you cannot just open this in the browser to launch the app. Instead, the EHR will open this page and will pass some URL parameters. Without those parameters the launch will not work. Alternatively, you can use a launcher like http://launch.smarthealthit.org/ to pretend that you are the EHR and launch your app with sample patients data.

At this point you need to make sure your app is hosted on an HTTP server. If you have NodeJS installed, the simplest way to host these files would be to cd into your project folder (where the launch.html is saved) and run npx http-server.

Step 3.2. Create the index.html page

After the launch sequence is started we will be redirected to the EHR where we may have to authorize the launch and select a patient (depending on the requested scopes). Once that is complete, the EHR will redirect back to our site using the redirectUri option provided above. Once we are there, we have to complete the authorization flow, which the client library will do for us. Then we can start making FHIR request and implement our business logic. Here is an example of simple app that renders as JSON the current patient and his/her medications (if any):

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>Example SMART App</title>

<script src="https://cdn.jsdelivr.net/npm/fhirclient/build/fhir-client.js"></script>

<style>

#patient, #meds {

font-family: Monaco, monospace;

white-space: pre;

font-size: 13px;

height: 30vh;

overflow: scroll;

border: 1px solid #CCC;

}

</style>

</head>

<body>

<h4>Current Patient</h4>

<div id="patient">Loading...</div>

<br/>

<h4>Medications</h4>

<div id="meds">Loading...</div>

<script type="text/javascript">

FHIR.oauth2.ready().then(function(client) {

// Render the current patient (or any error)

client.patient.read().then(

function(pt) {

document.getElementById("patient").innerText = JSON.stringify(pt, null, 4);

},

function(error) {

document.getElementById("patient").innerText = error.stack;

}

);

// Get MedicationRequests for the selected patient

client.request("/MedicationRequest?patient=" + client.patient.id, {

resolveReferences: [ "medicationReference" ],

graph: true

})

// Reject if no MedicationRequests are found

.then(function(data) {

if (!data.entry || !data.entry.length) {

throw new Error("No medications found for the selected patient");

}

return data.entry;

})

// Render the current patient's medications (or any error)

.then(

function(meds) {

document.getElementById("meds").innerText = JSON.stringify(meds, null, 4);

},

function(error) {

document.getElementById("meds").innerText = error.stack;

}

);

}).catch(console.error);

</script>

</body>

</html>

Now you can launch this against different EHRs or sandboxes. To do so, you will have to “register” the app, get it’s client ID and put it in the clientId field in your launch.html. The easiest way to try this would be to use the SMART sandbox at http://launch.smarthealthit.org which does not even require you to register the app. Furthermore, if your app is running on http://127.0.0.1:8080/, then you can launch it simply by clicking [HERE](http://127.0.0.1:8080/launch.html?launch=eyJhIjoiMSJ9&iss=https%3A%2F%2Flaunch.smarthealthit.org%2Fv%2Fr4%2Ffhir).